International Institute of Information Technology Bangalore

System Design with FPGA(VLS 505)

Project Report

On

"I**mplementation of TIFF image compression algorithm using Vitis HLS** "

Submitted by

Priyanka Agarwal (PH2023504)

Arun M (MS2024004)

Chandan Kumar N S (MS2024007)

Under the guidance of

Prof. Nanditha Rao

ABSTRACT

This report outlines the process of implementing image compression by utilizing the functionalities provided by the TIFF (Tagged Image File Format) library, specifically leveraging the `tiffcp.c` code from the LibTIFF project. The goal is to isolate the compression algorithm, subsequently undergo High-Level Synthesis (HLS), and apply performance optimizations aimed at reducing latency. This study investigates the feasibility of hardware-accelerated image processing, which is crucial in applications demanding high throughput and low latency.

Note:

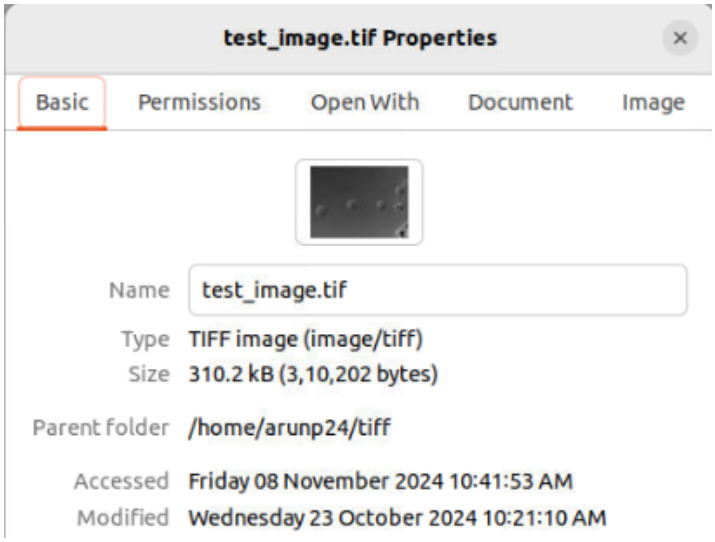
1. The report is divided into sections. Each section delves with the methods, issues and the corresponding outputs we have obtained.
2. A github link is provided where all the codes and pictures used for the project is added.
3. For easy evaluation, the codes have been added with a comment line (within the file) specifying the section for which it matches.

Github Link: https://github.com/ArunM2402/TIFF

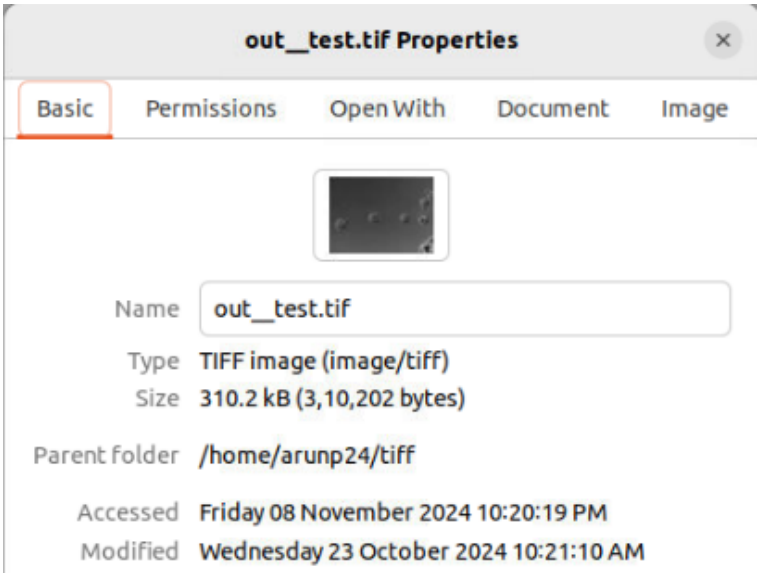
SECTION 1:
Understanding Tiff format
The **TIFF** (Tagged Image File Format) is a flexible and widely used image format that supports various color depths, multiple layers, and both lossy and lossless compression methods. Here, the header is a crucial part of the file structure. It contains metadata and information like Byte Order, Version Number, IFD etc which are necessary for correctly interpreting the image data stored in the file. The IFD contains the actual tags (metadata) for the image, such as image width, height, and compression type.

SECTION 2:
In this section, we have tried reading a TIFF image and writing it back without any compression. This was done to verify the read function and the write function of the C code.

The size of the input image is shown below:



The size of the otuput image is :

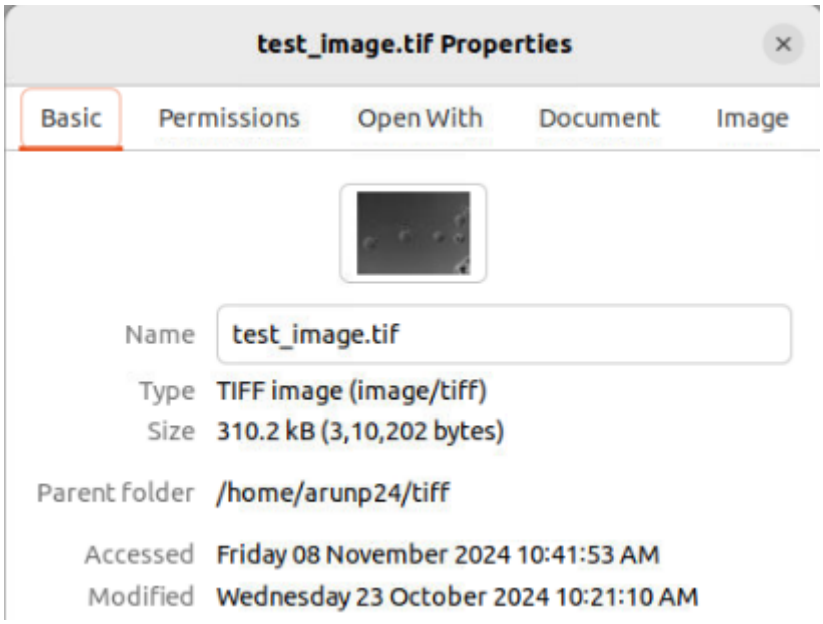


The sizes match hence the code is verified.

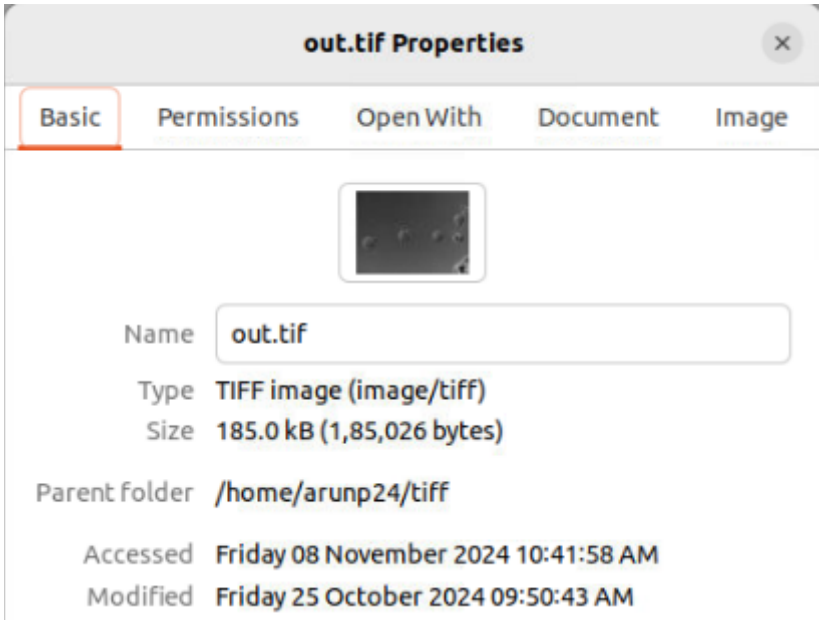
SECTION 3:

In this section, we have tried reading a TIFF image and writing it back after compression. The compression is done using built-in function(COMPRESSION_DEFLATE). The code was able to achieve % of compression.

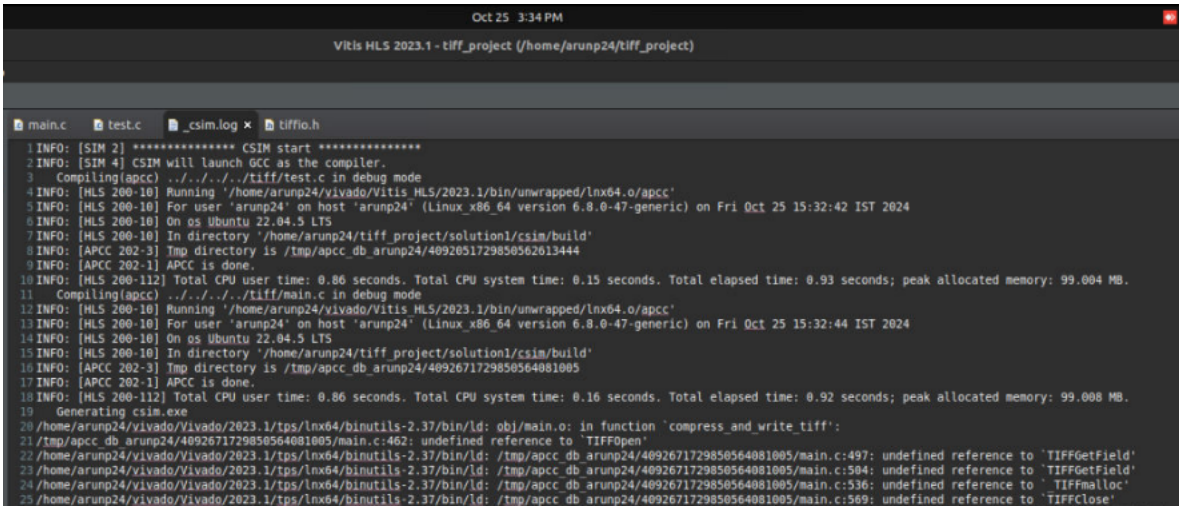
The size of the input image is:



The size of the output image is:

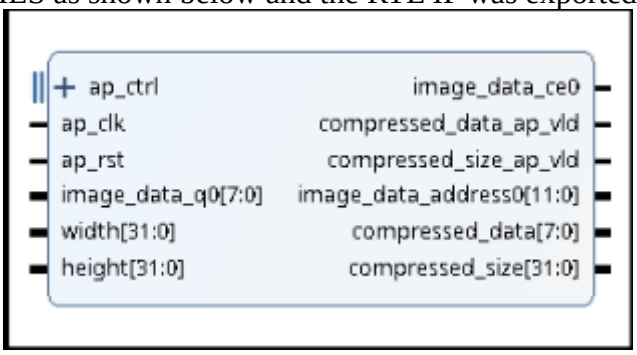


SECTION 4:
In this section, we try the code to go through HLS. However, it faced the following issue



This was because of the header files and built-in functions which we had used. HLS simulation and synthesis does not support the use of header files. Hence we switched to a code where we implemented a simple version of the compression technique aimed at mimicking the built-in function.

SECTION 5:
In this section, we try HLS again with the modified C code without using header files. This code was able to go through HLS as shown below and the RTL IP was exported.

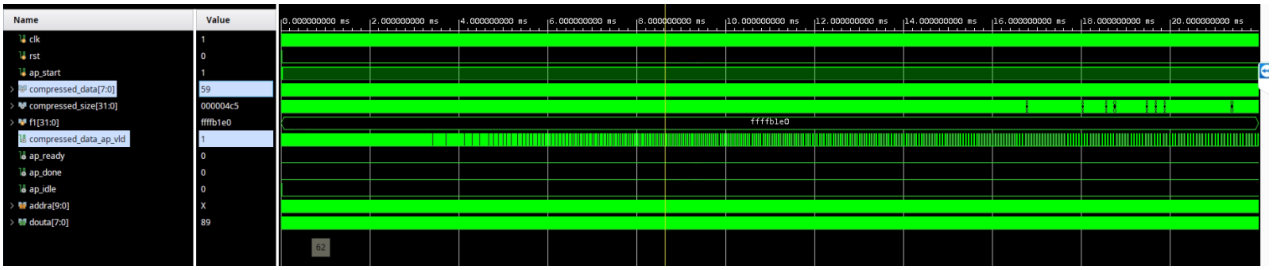


However, since reading and writing of images are not supported directly in Vivado, the pixel values of the image was extracted using a python code into a .coe file. This was inturn given to a blockram. The IP reads in values from the blockram, compressed the values and gives us the compressed data.

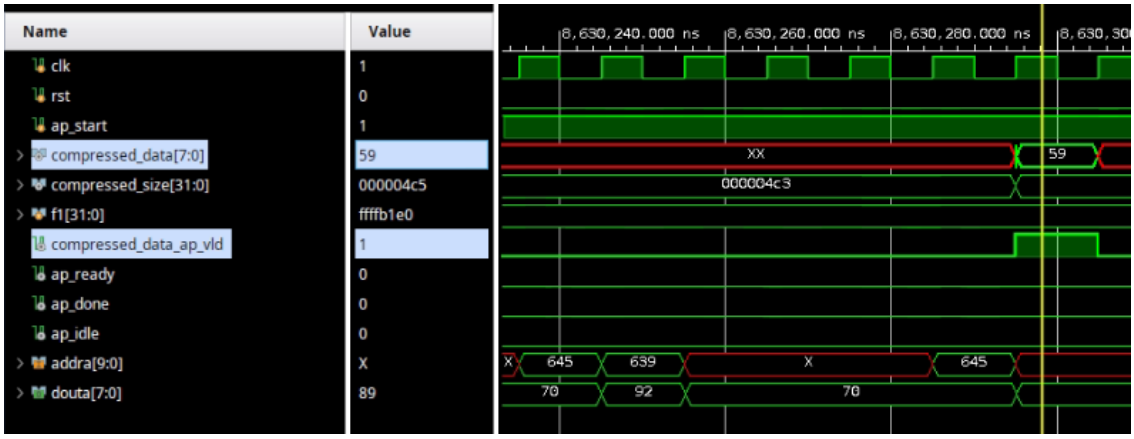
Since Vivado shows a maximum value error and causes segmentation fault, we have constrained ourselves to grayscale and 3 types of images.

a) 32 X 32

The results are as shown below:

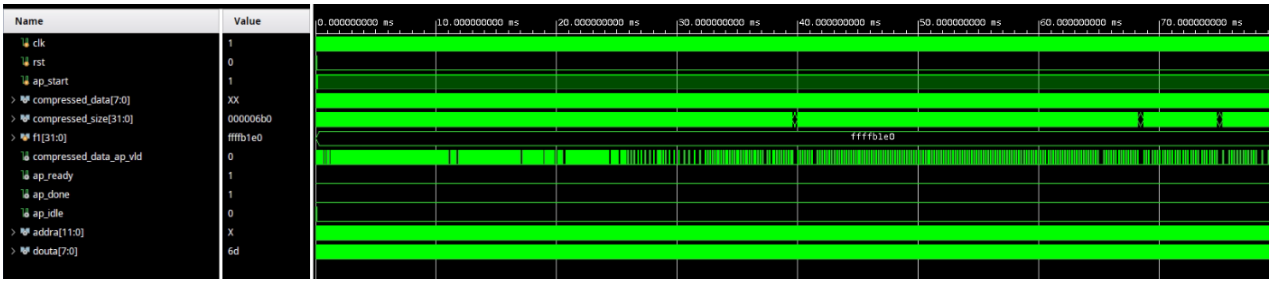


Since the above waveform is difficult to analyze, we have added a zoomed viewed of one pixel. Note that there would be compressed pixel value only when the ap_valid signal is high.

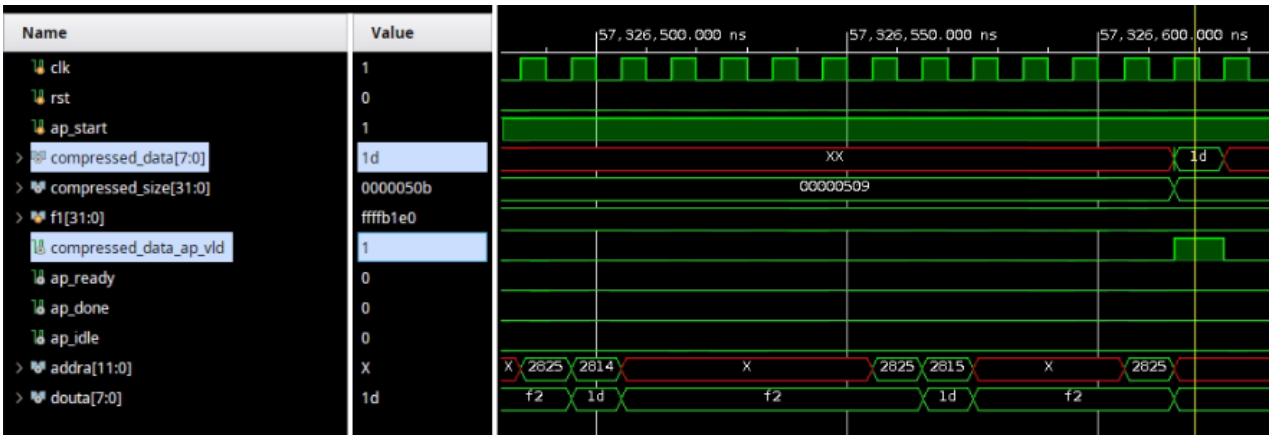


b) 64 X 64

The results are as shown below:

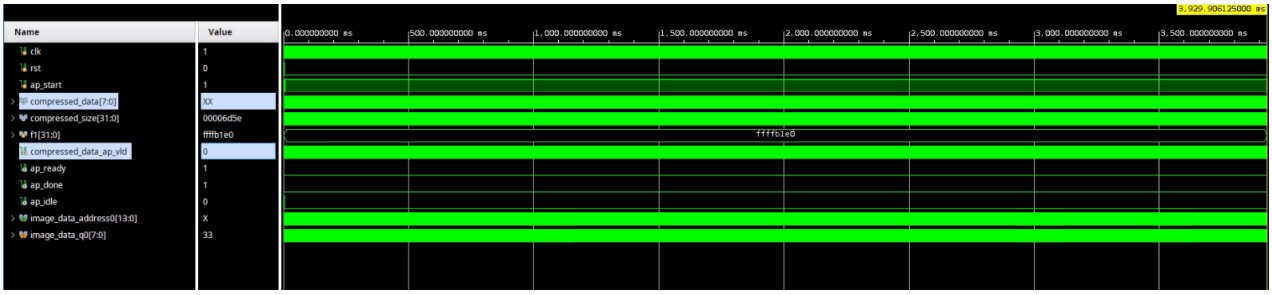


Since the above waveform is difficult to analyze, we have added a zoomed viewed of one pixel. Note that there would be compressed pixel value only when the ap_valid signal is high.

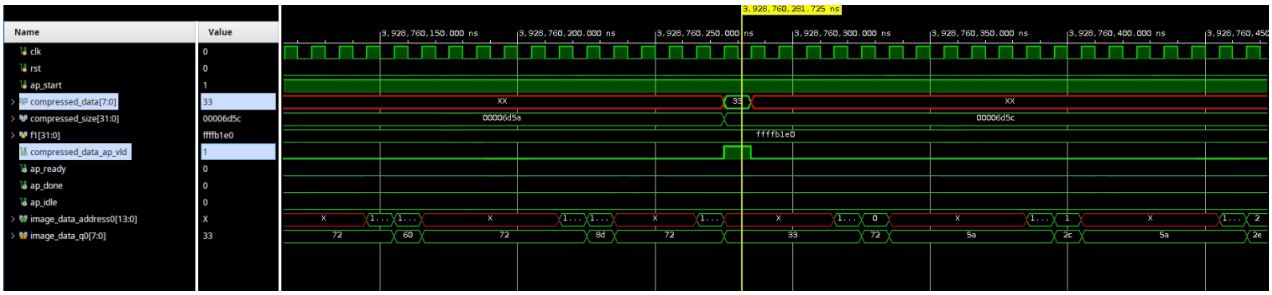


c) 128 X 128

The results are as shown below:



Since the above waveform is difficult to analyze, we have added a zoomed viewed of one pixel. Note that there would be compressed pixel value only when the ap_valid signal is high.



SECTION 6:

In HLS, we can improve the performance of the IP by using different pragmas. Different solutions are obtained using different pragmas and are compared as shown.

a) Without Pragmas

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
7	770	70.000 ns	7.700 us	7	770	no

Detail

Instance

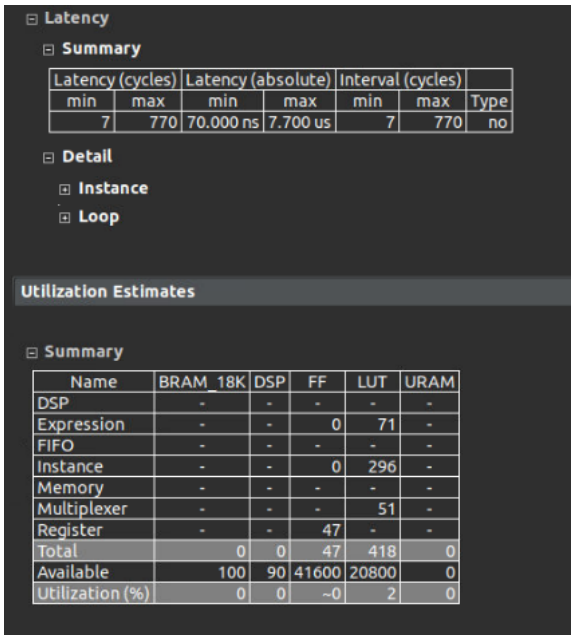
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	98	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	64	-
Register	-	-	55	-	-
Total	0	0	55	162	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	~0	~0	0

b) using array partitioning for image data and loop unroll factor 2



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
7	770	70.000 ns	7.700 us	7	770	no

Detail

Instance

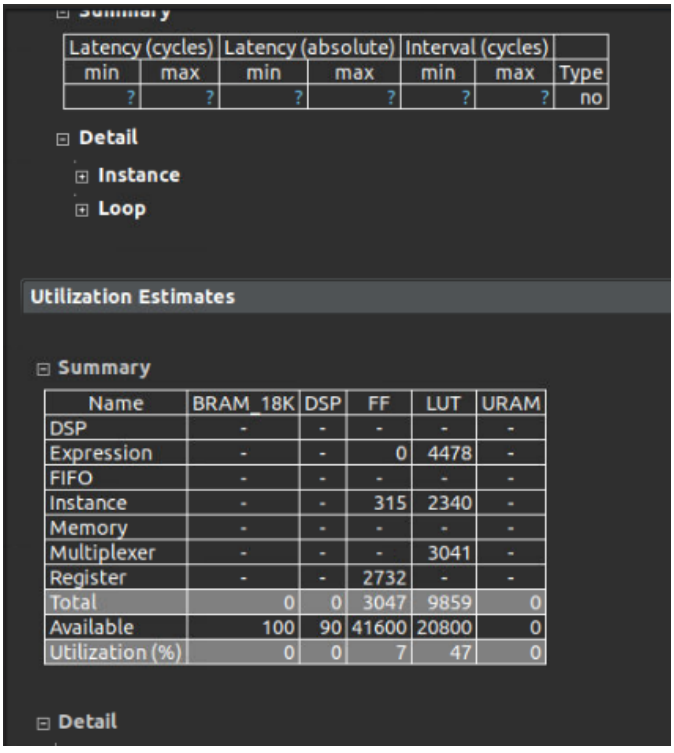
Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	71	-
FIFO	-	-	-	-	-
Instance	-	-	0	296	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	51	-
Register	-	-	47	-	-
Total	0	0	47	418	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	~0	2	0

c) using array partitioning for image data and loop unroll factor 10



Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	no

Detail

Instance

Loop

Utilization Estimates

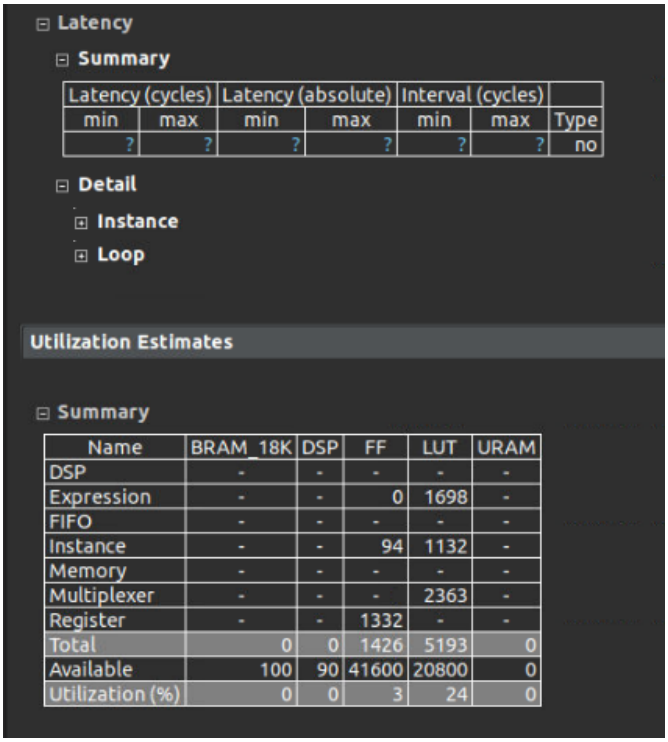
Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4478	-
FIFO	-	-	-	-	-
Instance	-	-	315	2340	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	3041	-
Register	-	-	2732	-	-
Total	0	0	3047	9859	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	7	47	0

Detail

Here, the latency is shown to be undeterministic as it requires more information about the signals of the IP which can be avaliable only after exporting and interfacing with blockram.

d) using array partitioning for image data,loop unroll factor 2 and performance pragma



Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
?	?	?	?	?	?	no

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	1698	-
FIFO	-	-	-	-	-
Instance	-	-	94	1132	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	2363	-
Register	-	-	1332	-	-
Total	0	0	1426	5193	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	3	24	0

REFERENCES

- https://gitlab.com/libtiff/libtiff/-/tree/master
- https://download.osgeo.org/libtiff/doc/TIFF6.pdf